VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FRAMEWORK PRO TVORBU A OVĚŘOVÁNÍ SPECIFIKAČNÍCH MODELŮ

FRAMEWORK FOR CREATION AND VERIFICATION OF SPECIFICATION MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠTĚPÁN MATALÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Matalík Štěpán**
Obor: Informační technologie
Téma: **Framework pro tvorbu a ověřování specifikačních modelů**
Framework for Creation and Verification of Specification Models
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s problematikou nástrojů pro podporu vývoje softwarových systémů. Prostudujte formalismus Objektově orientovaných Petriho sítí (OOPN), simulátor OOPN a jeho komunikační protokol.
2. Navrhněte architekturu nástroje, který umožňuje vytvářet a ověřovat specifikační modely softwarových systémů. Nástroj bude pracovat s modelovacím formalismem OOPN, bude umožňovat editaci modelů, komunikaci se simulátorem, ovládat simulaci a zobrazovat aktuální stav simulace. Nástroj musí umožnit export a import modelů.
3. Navržená architektura bude modulární, tj. bude otevřená rozšíření o další modelovací formalismy.
4. Navržený nástroj implementujte v jazyce Java.
5. Vytvořte manuál a sadu příkladů demonstrující možnosti nástroje.
6. Diskutujte dosažené výsledky a navrhněte možná rozšíření nástroje. Výsledky také prezentujte formou posteru.

Literatura:

- V. Janoušek: Modelování objektů Petriho sítěmi. Disertační práce. Brno, 1998.
- C. Larman: Applying UML and Patterns. Prentice Hall, 2005.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá objektově orientovanými Petriho sítěmi (OOPN) a tvorbou nástroje pro tvorbu a ověřování specifikačních modelů popsaných OOPN. V první části práce jsou popsány formalismy existujících typů Petriho sítí, včetně OOPN. Další část se zabývá tvorbou editačního nástroje, který také umožňuje spouštění simulačních modelů na vzdáleném serveru. Výstupem editoru je jednak grafická podoba Petriho sítě a také simulační model popsaný jazykem PNtalk, což je konkrétní implementace OOPN založená na jazyku Smalltalk. Simulační modely jsou spouštěny na serveru v prostředí Pharo. Na závěr práce jsou ukázány příklady konkrétních simulací a možná rozšíření nástroje.

Abstract

The thesis deals with Object Oriented Petri Nets (OOPN) and with developing of tool for creation and verification of specification models. In the first part of the thesis are described formalisms of existing Petri net types, including OOPN. Next part involves in creation of editing tool, that also allows triggering of simulation models on a remote server. Editor output is a graphic diagram of Petri net and also a simulation model described in PNtalk language, which is the implementation of OOPN based on Smalltalk language. Simulation models runs on the server in Pharo Smalltalk enviroment. At the end of the thesis are shown examples of appropriate simulations and possible tool extensions.

Klíčová slova

Petriho sítě, Java, Editor Petriho sítí, Objektově orientované Petriho sítě, PNtalk, IntelliJ IDEA, Pharo.

Keywords

Petri nets, Java, Petri net editor, Object oriented Petri nets, PNtalk, IntelliJ IDEA, Pharo.

Citace

MATALÍK, Štěpán. *Framework pro tvorbu a ověřování specifikačních modelů.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Kočí, Ph.D.

Framework pro tvorbu a ověřování specifikačních modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Štěpán Matalík
17. května 2018

Poděkování

Rád bych poděkoval za odborné a velmi cenné rady poskytnuté vedoucím bakalářské práce Ing. Radkem Kočím Ph.D.

Obsah

Kapitola 1

Úvod

Petriho sítě umožňují návrh, verifikaci a analýzu dynamických diskrétních systémů. Poskytují matematický formalismus, pomocí kterého lze vytvářet simulační modely obsahující nedeterminismus i paralelismus. Petriho síť v původní podobě, jak ji navrhl Carl Adam Petri, neobsahuje strukturovací mechanismy (funkce, třídy a objektovou orientaci), pomocí kterých lze snadno modelovat rozsáhlé systémy a tak je jejich použitelnost značně omezená na malé systémy. Také zde nemáme výhody objektově orientovaných jazyků, jako je znovupoužitelnost, zapouzdření, polymorfismus či snadná rozšířitelnost. Tento problém řeší zavedení dalších typů sítí, obzvláště pak vysokoúrovňových Petriho sítí v čele s OOPN. Systém OOPN již programátorovi poskytuje třídní systém, objekty a metody. Díky dostupnosti těchto prvků lze modelované systémy velmi dobře strukturovat. [6]

1.1 Cíl práce

Cílem práce je vytvořit nástroj pro tvorbu a ověřování specifikačních modelů, který bude pracovat s modely zapsanými v jazyce PNtalk. Tento nástroj by měl poskytovat uživateli editační i simulační funkcionalitu, tzn., že uživatel by měl být schopný za pomocí nástroje jednak modely vytvářet, editovat a ukládat, ale také nahrávat modely na server, na něm je spouštět a krokovat. Při vývoji by měl být kladen důraz na přívětivé a intuitivní grafické uživatelské rozhraní, které uživateli nebude překážkou, ale důvodem, proč si nástroj oblíbí. Dále je žádoucí, aby navržená architektura byla modulární, otevřená dalším modelovacím formalismům a dalším rozšíření.

1.2 Popis kapitol

V kapitole 2 se seznámíme s různými druhy Petriho sítí. Začneme se základním konceptem, na který navážeme nízkoúrovňovými a vysokoúrovňovými Petriho sítěmi. Popsán bude formalismus různých typů sítí i význam jednotlivých grafických prvků. V 3. kapitole se budeme věnovat funkcionálním Petriho sítím a zakomponování objektové orientace do Petriho sítí. V kapitole 4 bude představen jazyk PNtalk. Tento jazyk splňuje formalismus OOPN, proto bude použit pro spouštění vytvořených modelů na serveru.

V následující 5. kapitole je pozornost věnována již analýze našeho zadaného problému a analýze existujících aplikací pro modelování Petriho sítí. V kapitole 6 bude představen návrh grafického rozhraní i návrh architektury. Dále se v této kapitole budeme postupně věnovat jednotlivým částem aplikace zajímavých z hlediska implementace. Opomenut ne-

bude například popis konverze objektového modelu do jazyka XML pro ukládání a načítání modelů ze souboru nebo konverze do jazyka PNTalk pro odesílání vyhotovených simulačních modelů na server. Kapitola 7 obsahuje ukázky krokovaných příkladů vytvořených a simulovaných v našem editoru a v závěrečné 8. kapitole bude v první části zhodnocen výsledek z hlediska splnění požadavků a v druhé části se nastíní možná rozšíření programu v budoucnu.

Kapitola 2

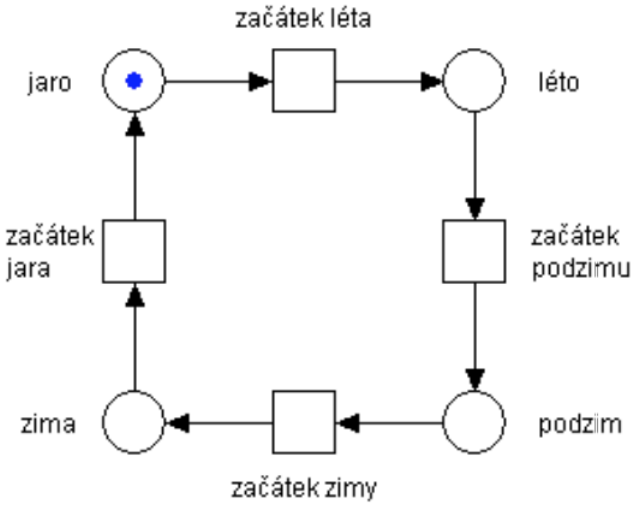
Petriho sítě

Tato kapitola nás uvede do základů Petriho sítí, které je třeba pochopit před tím, než se budeme věnovat objektově orientovaným Petriho sítím (dále jen OOPN). Petriho sítě jsou již známy od šedesátých let a za celou dobu vznikla řada nových typů, které vycházejí ze základních konceptů, my se budeme věnovat pouze těm, které mají pro tuto práci význam[7].

2.1 Základní koncept

Petriho síť je matematický formalismus, pomocí kterého se modelují diskrétní systémy. Vychází z konečných automatů a rozšiřují je o prvek zachycení přechodu mezi dvěma stavy. Podmínku provedení přechodu představuje stav, který může být dekomponován na více stavů. Z toho vyplývá, že přechod je proveden až po splnění všech dílčích podmínek, v grafickém zobrazení to znamená, že do přechodu mohou vést hrany z více stavů [6].

Zmíněná dekompozice je v praxi hodně využívána, s její pomocí můžeme lehce modelovat paralelismus a nedeterminismus, hojně se využívá k návrhu linek, systémů a k testování.



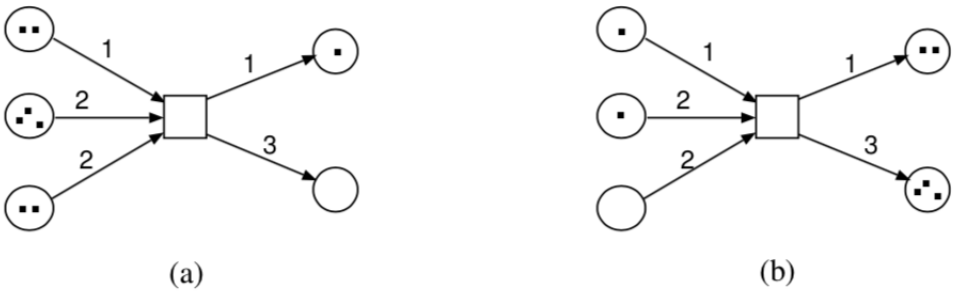
Obrázek 2.1: Příklad C/E Petriho sítě [6].

2.2 C/E Petriho sítě

Označení C/E Petriho síťě vzniklo podle prvků, z kterých se tento typ skládá - podmínka (condition) a událost (event). Podmínky jsou graficky vyobrazeny jako kruhy, události představují obdelníky nebo čtverce, jak můžete vidět na obrázku 2.1. Pokud je podmínka splněna, obsahuje tzv. token, ten je značen jako tečka. Podmínky a události jsou propojeny hranou, graficky značenou jako šipka. Ta nesmí spojovat stejné prvky (dvě události nebo dvě podmínky), ale prvky rozdílné (podmínku a událost) v kterémkoliv směru. Hrana směřující z události do podmínky představuje výstupní podmínku události, hrana směřující z podmínky do události představuje vstupní podmínku události [6].

2.3 P/T Petriho sítě

Tato síť vychází z konceptu C/E Petriho sítě. Namísto podmínek a událostí tvoří kromě hran její základní prvky ještě *místa* (places) a *přechody* (transitions). Místa se od podmínek liší kapacitou, která udává maximální počet tokenů v místě. Kapacita může být také nekonečná, v tom případě se kapacita neuvádí žádná. Z toho vyplývá, že implicitně je předpokládána nekonečná kapacita. Hrany v tomto typu sítě zastávají stejnou funkci, jsou jen rozšířeny o *váhu*, která uvádí, kolik tokenů je odebráno z příslušného místa při provedení přechodu. Příklad takové sítě před a po provedení přechodu můžeme vidět na obrázku 2.2. Grafické značení se od C/E sítí neliší. Na vztah C/E a P/T sítí lze nahlížet také tak, že C/E Petriho síť je speciálním případem P/T Petriho sítě [2].

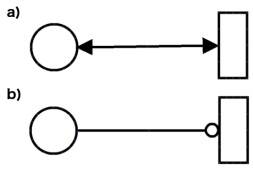


Obrázek 2.2: Příklad provedení přechodu v P/T Petriho síti [3].

P/T Petriho sítě jsou základem pro další rozšíření. V následujících podkapitolách se zaměříme pouze na ty, jejichž znalost má význam pro pochopení problematiky probírané v dalších kapitolách.

2.3.1 Testovací hrana

Testovací hrana se značí obousměrnou šipkou, jak můžeme vidět na obrázku 2.3 a). Její sémantika je ekvivalentní ke dvěma hranám, přitom jedna vede z místa do přechodu a druhá obráceně. Tohle zapříčiní, že při provedení přechodu nezmizí token ze vstupního místa, které je propojeno s přechodem testovací hranou.



Obrázek 2.3: a) testovací hrana, b) inhibitor.

2.3.2 Inhibiční hrana

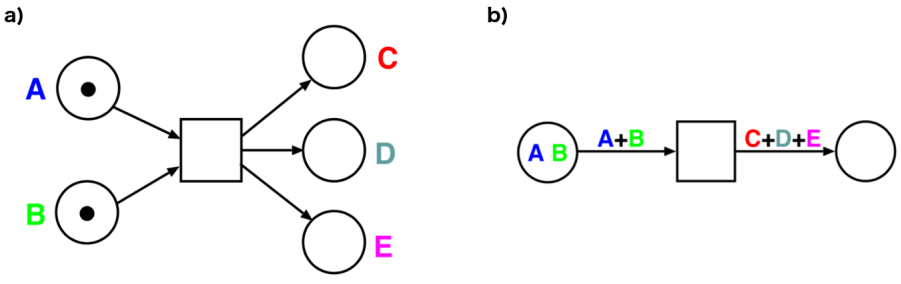
Rozšíření, kterému se zkráceně říká inhibitor, je dalším speciálním případem hrany. Inhibitor přidává k proveditelnosti hrany podmínku, která vyžaduje, aby bylo v místě spojeném inhibitorem k přechodu méně tokenů, než je ohodnocení inhibitoru [3]. Dá se tedy říci, že inhibitor je opak obyčejných hran s ohodnocením.

Existují ještě další rozšíření, jako Petriho síť s pravděpodobností nebo časovaná Petriho síť. Jelikož tato rozšíření nejsou přímo podporována v jazyce PNtalk, nebudeme se jimi zabývat.

2.4 Barvené Petriho sítě

Tento typ patří mezi vysokoúrovňové Petriho sítě. Přestože P/T sítí se všemi rozšířeními lze modelovat vše, co lze vyjádřit algoritmem, v některých případech není jejich použití příliš praktické, protože některé jednoduché situace se modelují příliš složitě. Modelování složitějších systémů nízkoúrovňovou Petriho sítí lze tak přirovnat k programování složitější aplikací v assembleru [3].

Barvené sítě vychází z konceptu P/T sítí, do něj nově zavádí typy tokenů, které se v praxi odlišují právě barvami. Každé místo má svou množinu barev, která určuje, jaké barvy tokenů dané místo přijímá. Barvy tak zavádějí do P/T sítě datové typy. Součástí přechodu může být podmínka přechodu, neboli stráž, ta se vyhodnocuje před provedením přechodu. Hrany mohou obsahovat hranový výraz. Stráž i hranový výraz jsou tvořeny z konstant a proměnných. Výsledkem po vyhodnocení hranového výrazu je multimnožina tokenů, která obsahuje tokeny tříd, které náleží místu připojeném touto hranou [2].



Obrázek 2.4: a) P/T Petriho síť, b) ekvivalentní barvená Petriho síť [6].

Na obrázku 2.4 b) vidíme barvenou Petriho síť, která je ekvivalentní k síti na obrázku 2.4 a). Můžeme si všimnout výše zmíněných faktů, že prostorová náročnost barvené sítě je mnohem menší, ale také že algebraická analýza barvené sítě je složitější, než analýza P/T sítě.

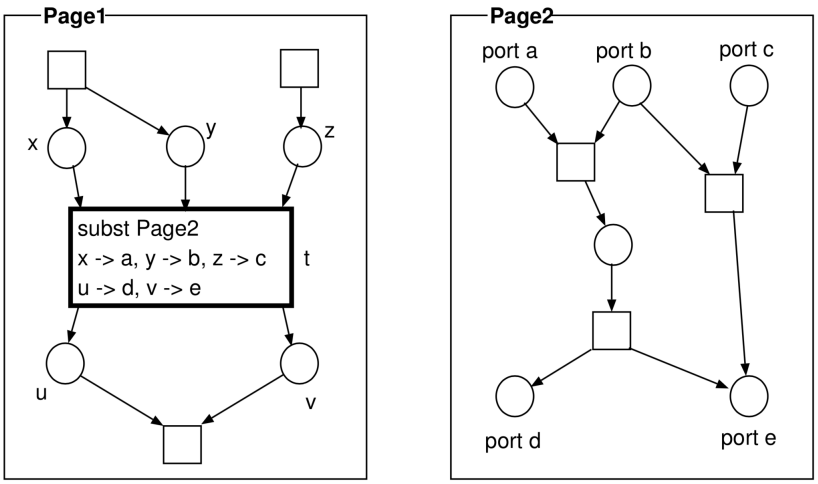
Pomocí barvených sítí můžeme jednodušeji modelovat složitější výpočty. Barvené sítě si s sebou nesou jistou nevýhodu, a to že jejich algebraická analýza je značně obtížnější než analýza obyčejné P/T sítě, avšak v každém případě je možná. To platí obecně pro všechny vysokoúrovňové Petriho sítě [3]. V praxi se často používají písmenné pojmenování tokenů namísto barev.

2.5 Hierarchické Petriho sítě

Hierarchické sítě se používají na modelování systémů, které obsahují více subsystémů s totožnou strukturou. Pro použití hierarchického přístupu je třeba, aby se počet a struktura subsystémů neměnila během života sítě [3].

Síť je tvořena množinou stránek. Ty jsou pojmenované, aby bylo možno se na stránky odkazovat (podobně jako jména funkcí v programování). Každá stránka definuje svou vlastní síť a porty, to jsou vstupy a výstupy sítě stránky. Porty představují přechody nebo místa, záleží na typu substituční stránky připojené k hierarchicky nadřazené instanci:

- Substituční přechod - stránka definuje místa jako porty.
- Substituční místo - stránka definuje přechody jako porty.



Obrázek 2.5: Hierarchická Petriho síť se substitučním přechodem [3].

Na obrázku 2.5 si všimněme, že v hierarchicky nadřazené síti, z které je substituční přechod volán, se musí definovat navázání míst na porty pomocí symbolu šipky. U substitučního místa se musí analogicky definovat přechody na porty.

Kapitola 3

Objektově orientované Petriho sítě

OOPN jsou dalším druhem vysokoúrovňových Petriho sítí. Přidáním objektové orientace k funkcionálnímu konceptu Petriho sítí získáme nástroj, pomocí kterého lze pohodlně modelovat rozsáhlé systémy. V OOPN můžeme vidět i některé prvky z hierarchických a barvených sítí. Díky zavedení třídního modelu a metod získáváme skutečně efektivní znovupoužitelnost, rozšířitelnost i větší přehlednost v celé struktuře programu.

3.1 Funkcionální Petriho sítě

Funkcionální Petriho sítě můžeme brát za mezikrok k OOPN. Jsou zde zavedeny funkce, které opět o něco ulehčují modelování, o dobře použitelném přístupu však mluvit nelze. Jednak se funkcionální modely příliš nevyužívají a jednak jsou funkce nedeterministické, což není pro nás žádoucí [7].

3.1.1 Funkce a jejich volání

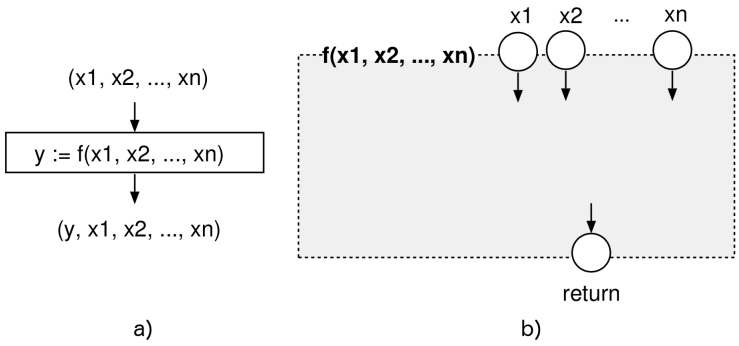
Funkci představuje hierarchicky vnořená Petriho síť. Při jejím zavolání se vytvoří nová instance sítě a naplní se její parametrová místa (ta nejsou povinná) příslušnými značkami vstupních míst. Funkce se následně začne provádět paralelně s hierarchicky nadřazenou sítí. Každá funkce musí obsahovat návratové místo return. Jakmile dojde k umístění značky do tohoto místa, značka se přiřadí do příslušné proměnné v akci přechodu z kterého funkce byla invokovaná a instance funkce se zruší.

Ve funkcionálních Petriho sítích jsou funkce volány z tzv. invokačního přechodu, tj. přechod, z kterého se volají funkce nebo procedury. Syntaxe volání funkce z invokačního přechodu je stejná jako v inskripčním jazyce[1] Petriho sítě, aby byl systém s tímto jazykem kompatibilní. Volání funkce je značeno klasicky identifikátorem funkce, za kterým jsou v závorkách řárkou oddělené parametry, přitom výsledek funkce se přiřazuje do proměnné notací „:=“. Příklad takového volání funkce z akce přechodu můžeme vidět na obrázku 3.1 Díky tomu, že je syntaxe stejná pro volání funkce v inskripčním jazyce a Petriho sítích, není třeba dodatečně specifikovat propojení vstupních míst a portů funkce [3].

3.1.2 Funkce s vedlejším efektem

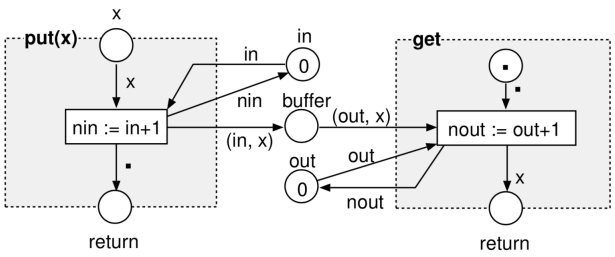
Tento modelovací prvek je důležitý v OOPN. Umožňuje přechodům z instance sítě přistupovat ke globálním místům a modifikovat tak jejich obsah [3]. V OOPN se tohle používá

[1] Jazyk blízký běžnému programovacímu jazyku, který se používá k popisu Petriho sítí.



Obrázek 3.1: a) volání funkce v akci přechodu b) struktura funkce [3].

u přistupování k místům objektové sítě z instancí sítí metod. To umožňuje efektivnější provázání funkcí s objektovou sítí.



Obrázek 3.2: a) Provázání funkcí s globálními místy [3].

3.2 Objektová orientace v Petriho sítích

V OOPN přináší objektová orientace výhody jako u jiných objektově orientovaných jazyků, dědičnost, zapouzdření a polymorfismus. Předpokládá se znalost těchto pojmů v kontextu programování, protože v následujících podkapitolách bude vysvětlena aplikace těchto principů na Petriho sítě.

3.2.1 Zapouzdření

Zapouzdření je realizováno tak, že funkce nazveme metodami a jejich množinu spolu s množinou míst, která metody sdílí nazveme objektem. Místa tak tvoří atributy objektu. Poté umožníme dynamickou instanciaci takto vytvořených objektů. Celá takový struktura se nazývá třídou dynamicky instanciovatelných objektů.

V praxi můžeme vytvářet nové objekty, které jsou instancemi třídy. Každému objektu se přidělí unikátní identita a vlastní paměť pro atributy objektu. Po přijmutí zprávy objektem se invokuje instance sítě odpovídající metody, která má přístup k atributům objektu, který zprávu obdržel. K atributům jiného objektu včetně atributům jiného objektu stejné třídy nemůže instance metody přistupovat [3].

3.2.2 Polymorfismus

V OOPN se koncept polymorfismu nijak neliší od obvyklého použití polymorfismu v jiných programovacích jazycích. Objekt pošle jinému objektu zprávu složeného ze jména třídy, které představuje adresáta zprávy a jména metody, to představuje selektor zprávy. Adresát a selektor se oddělí tečkou. Za tento identifikátor metody se do závorek zapíší v příslušném pořadí argumenty.

Polymorfismus nám umožňuje fakt podílení adresáta (jména třídy) na identifikaci instance metody. Díky tomu může být zpráva poslána objektům různých tříd a v rámci těchto tříd jsou pro totéž jméno implementovány různé metody [3].

3.2.3 Dědičnost

Poslední důležitou vlastností, kterou by objektově orientované jazyky měly mít je dědičnost. Ta umožňuje specifikovat nové třídy, jako podtřídu již existující třídy. Od té dědí podtřída objektovou síť i metody, specifikují se pouze rozdíly. OOPN umožňují použít jen koncept jednoduché dědičnosti, v kterém není povoleno dědit z více tříd současně.

Metody mohou být zděděny bez dalších modifikací, nebo může být metoda nahrazena novou definicí. Zděděné objektové sítě mohou být ponechány bez další modifikace, mohou být k nim přidány další uzly, nebo již existující uzly mohou být předefinovány [3].

3.3 Struktura třídy v OOPN

Struktura OOPN je tvořena množinou tříd. Ta je uspořádána hierarchicky podle vztahu dědičnosti. Jedna třída je prohlášena za třídu počáteční, která je při spuštění sítě implicitně instanciována. Třída je tvořena objektovou sítí, množinou metod a množinou synchronních portů.

3.3.1 Objektová síť

Popisuje strukturu a stav instancí dané třídy. Dále popisuje také aktivitu atributů a jejich chování. V každé třídě musí existovat právě jedna objektová síť [6].

3.3.2 Metoda

Metody jsou reprezentovány funkcemi, které jsou vázány k jedné třídě. ty můžeme volat z jakéhokoliv přechodu. Síť metody vychází z objektové sítě, navíc může obsahovat volitelné parametry, představující místa, do kterých se při invokaci metody uloží parametry zprávy. Dále každá síť metody musí, na rozdíl od parametrů, povinně obsahovat místo return, ve kterém se očekává po ukončení provádění metody návratová hodnota. Metody využívají také koncept funkce s vedlejším efektem, který byl popsán v sekci 3.1.2 [3].

3.3.3 Predikát

Predikát je prvek, který představuje nástroj pro synchronní komunikaci. Umožňuje provedení prvku v závislosti na jiném objektu zavoláním predikátu stejným způsobem, jakoby se volala metoda. Predikát je k síti objektu připojen pouze testovacími hranami, tzn. že má přístup k místům objektu, ale pouze nedestruktivní. Hrany (popř. jejich výrazy) se vyhodnotí a vrátí se výsledek predikátu do akce přechodu z kterého byl predikát volán [3].

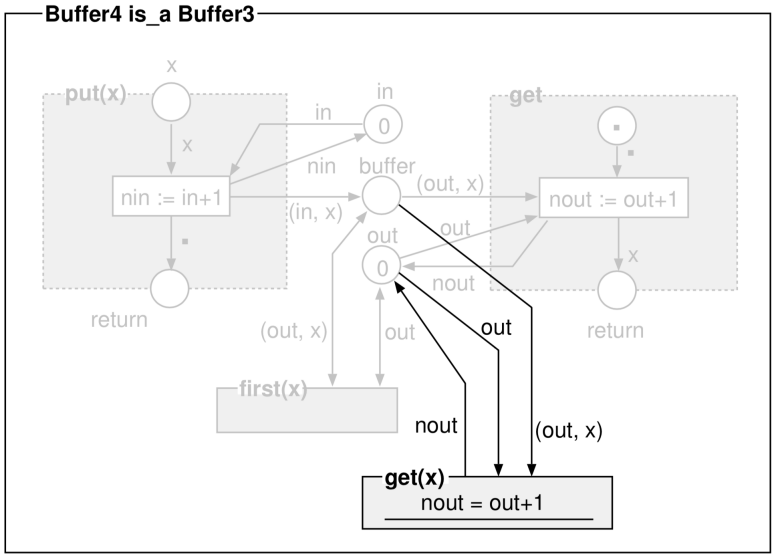
3.3.4 Synchronní port

Dalším prvkem pro synchronní komunikaci je synchronní port. Je to modifikace predikátu, která umožňuje měnit obsah objektové sítě, takže není připojený k objektové síti testovacími hranami ale hranami obyčejnými. Synchronní port může mít dále definovanou stráž, avšak nemůže mít definovanou akci jako přechod.

K synchronnímu portu se připojuje, podobně jako u sítě metody, vzor zprávy na kterou reaguje. Pomocí tohoto vzoru tak může být synchronní port volán z jakéhokoliv přechodu. Testování stráže volajícího přechodu nezapříčiní změnu stavu systému, uplatní se jen test proveditelnosti synchronního portu stejně jako u přechodu [3].

3.3.5 Negativní predikát

Negativní predikát je odvozen se synchronního portu. Zastává stejnou funkci s tím rozdílem, že má obrácenou sémantiku. Tzn. jestliže může být proveden negativní predikát, přechod, z kterého se predikát volá, proveden být nemůže a obráceně [5].



Obrázek 3.3: Příklad třídy v OOPN [3].

Na obrázku 3.3 můžeme vidět příklad třídy Buffer4, která dědí ze třídy Buffer3 objektovou síť, metody put(x), get a predikát first(x) a synchronní port get(x), který obsahuje stráž přechodu.

Kapitola 4

PNtalk

Jazyk PNtalk začal vyvíjet doc. Vladimír Janoušek na FIT VUT v roce 1993, aby přivedl vysokoúrovňové Petriho sítě blíže k programování [4]. PNtalk vychází z OOPN a můžeme jej prohlásit za jeho implementaci. Liší se pouze drobnými syntaktickými odlišnostmi, které mohou místy připomínat jazyk Smalltalk, právě proto jím bylo inspirováno jméno jazyku PNtalk. Tento jazyk jednak upřesňuje skutečnosti, které OOPN definuje jen volně, ale také rozšiřuje koncept OOPN o další funkcionality, které zjednodušují syntaxi jazyka.

4.1 Jazyk PNtalk

Na začátku si uvedeme syntaxi popisu prvků sítě (inskripčního jazyka), poté si popíšeme strukturu sítí, tříd a dědičnosti.

4.1.1 Termy

Nejjednodušším výrazem v PNtalku jsou termy. Představují objekty PNtalku.

- Literály. Jsou to primitivní objekty PNtalku. Patří zde čísla - literály složené z číslic, patří zde i záporná a desetinná, znaky - reprezentují prvky abecedy, jejich literál se uvozuje symbolem „$“, řetězce zapsané v uvozovkách, symboly - objekty používané jako jména, posloupnost znaků uvozena symbolem „#“, Booleovské konstanty - reprezentují identifikátory true a false a nedefinované objekty reprezentované identifikátorem nil (dá se analogicky přirovnat k identifikátoru null známého z většiny programovacích jazyků).
- Proměnné. Ty mohou během běhu programu reprezentovat různé objekty. Představují je identifikátory, které musí začínat malým písmenem. Jsou programově ovlivnitelné.
- Pseudoproměnné. Jde o dvě speciální proměnné self a super, jejichž hodnota závisí na aktuálním kontextu. Nejsou programově ovlivnitelné.
- Jména tříd. Jde o identifikátory s velkým počátečním písmenem. Jsou to konstanty, které se za běhu programu nemění.

[3]

4.1.2 Zprávy

Zpráva je výraz, který může být součástí akce i stráže přechodu. S vyjimkou termu má zpráva vždy syntaxi <adresát> <zpráva>. Zpráva se skládá ze selektoru a případně z argumentů zprávy. Rozlišujeme 3 druhy selektorů podle jejich tvaru:

- Unární zpráva. Zpráva je bez argumentů a její selektor je identifikátor (např. C new).
- Binární zpráva. Selektor zprávy je jeden z aritmetických, logických nebo speciálních operátorů. Po něm následuje jeden argument (např. u zprávy „4 / 2“ je „4“ adresátem zprávy „/ 2“, přičemž „/“ je selektor zprávy a „2“ je její argument).
- Zpráva s klíčovými slovy. Klíčové slovo je identifikátor zakončený dvojtečkou (např. „anObject at: 1 put: #e“, „anObject“ představuje identifikátor zprávy „at: 1 put: #e“, „at:“ a „put:“ jsou její selektory, „1“ a „#e“ její argumenty).

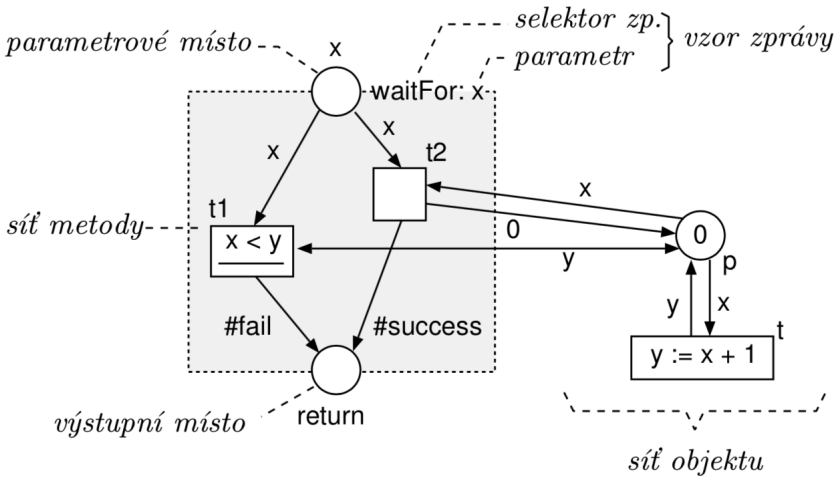
[3]

Všechny zprávy se vyhodnocují zleva doprava, pořadí vyhodnocování jde ale upravit použitím závorek. Detailnější popis sémantiky zpráv najdeme v popisu jazyka Smalltalk [1].

4.1.3 Sítě

I v PNtalku tvoří síť místa a přechody propojené hranami. Každá síť musí být součástí nějaké specifikace třídy objektu, existence samostatné sítě bez příslušnosti k třídě není v jazyce PNtalk možná. V PNtalku máme dva typy sítí:

- Síť objektu. Reprezentuje přechody a místa (atributy) objektu třídy a jeho vlastní aktivitu [3]. Je nezbytné, aby byla vytvořena právě jedna objektová síť v každé třídě. Příklad sítě objektu je na obrázku 4.1 vpravo.

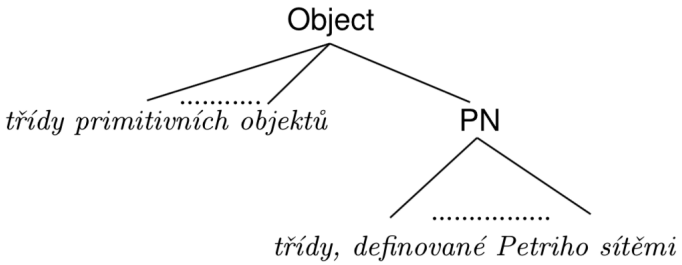


Obrázek 4.1: Síť metody (vlevo) a síť objektu (vpravo) v jazyce [3].

- Síť metody. Definuje chování objektu na příchozí zprávu. Síť metody tvoří stejně jako objektovou síť místa a přechody spojená hranami, rozdílem je možnost vytvořit libovolný počet metod v jedné třídě. Síť je obsahuje oproti síti metody vstupní místa a návratové místo return stejně jako v OOPN, navíc ale metoda obsahuje vzor zprávy, po jejímž přijetí objektem se dynamicky vytvoří instance metody a začne se provádět. Jména formálních parametrů ve vzoru zprávy musí odpovídat jménům parametrových míst v síti metody. Síť metody také může využávat již zmíněný koncept metody s vedlejším efektem, který se implementuje propojením přechodu sítě metody s místem sítě objektu. Příklad sítě metody je na obrázku 4.1 vlevo. [3].

4.1.4 Třídy a dědičnost

Model systému v PNtalku tvoří množina tříd, každá třída je tvořena jednou objektovou sítí, množinou sítí metod, synchronních portů a konstruktorů. Jedna třída je prohlášena za třídu počáteční, v praxi to znamená, že při startu programu se implicitně vytvoří její instance [3].



Obrázek 4.2: Hierarchie tříd v PNtalku [3].

Každá třída musí dědit právě z jedné třídy, třídy na vrcholu hierarchie dědičnosti v našem programu dědí z abstraktní třídy PN, jak můžeme vidět na obrázku 4.2. Třída od své nadtřídy podědí celou strukturu objektové sítě, ale taky všechny množiny sítí metod, synchronních portů i konstruktorů.

Metody, synchronní porty a konstruktory lze předefinovat jen úplně novou definicí, je však třeba zachovat stejný selektor zprávy. U objektových sítí můžeme navíc předefinovat jen její části. V OOPN hrany náleží přechodům, ke kterým jsou navázány, z toho vyplývá, že redefinujeme-li přechod, musíme redefinovat i hrany na něj navázané a redefinujeme-li místo, ponecháme již navázané hrany podděděné z nadtřídy [3].

Kapitola 5

Analýza

V rané fázi vývoje před samotnou implementací nástroje proběhla analýza. Ta zahrnovala pochopení konceptu OOPN a seznámení se s jazykem PNtalk. Další částí analýzy bylo prozkoumání již existujících aplikací pro tvorbu různých typů Petriho sítí a poslední část byla analýza vývojových nástrojů.

5.1 Analýza problému

V první fázi bylo třeba pochopit objektově orientované Petriho sítě a koncept jazyka PNtalk. V obou případech lze čerpat z disertační práce doc. Janouška [3] a to především z kapitoly 5 pro pochopení OOPN a kapitoly 6 pro systém PNtalk. Jelikož simulační část probíhá na serveru, není třeba implementovat interpret jazyka PNtalk a tak není nutné se zaobírat do větší hloubky syntaxí a sémantikou všech příkazů a operátorů jazyka.

Aplikace by měla umožňovat snadné modelování, proto je třeba věnovat pozornost obzvláště návrhu GUI[1] spjatém s modelováním objektů, ale také objektovému návrhu uložených dat, aby bylo jednoduché data měnit při zachování jejich konzistence. Dále je žádoucí, aby architektura programu byla modulární a rozšiřitelná o další rozšíření bez větších zásahů do již existujícího kódu.

5.2 Existujících programy pro modelování Petriho sítí

Pro získání většího přehledu o možnostech grafického rozhraní v modelovacím nástroji bylo vyzkoušeno pár již existujících řešení pro modelování Petriho sítí různých druhů. V této podkapitole budou tato řešení představena a budou shrnuty jejich výhody a nevýhody.

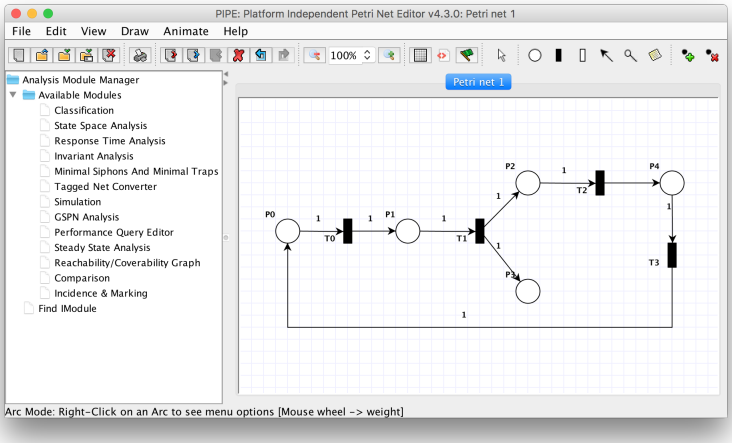
5.2.1 PIPE editor

PIPE editor[2] se začal vyvíjet v roce 2006 v Anglii v jazyce Java. Modelovací plocha je tvořena mřížkou, na kterou se objekty přichytávají, snadno tak lze objekty uspořádat. Vkládání prvků se provádí jejich výběrem v liště, jejich editace pak probíhá skrze kontextovou nabídku otevřenou nad příslušným prvkem. Aplikace mj. obsahuje také nástroje pro analýzu simulací a rozsáhlé statistiky.

[1] Graphics user interface - Grafické uživatelské rozhraní

[2] Platform Independent Petrin net Editor - Platformě nezávislý editor Petriho sítí

GUI je intuitivní a tak tvorba sítí v tomto nástroji probíhala hladce (dokud několikrát nenastala chyba projevující se nemožností šipku navázat na jakýkoliv prvek). Nástroj umožňuje také vytvoření lomené hrany (šipky), tu uživatel ocení obzvláště u modelování složitějších simulací. Přidání nového bodu, v kterém se hrana zalomí probíhá také přes kontextovou nabídku, avšak efektivnější metodou by byla možnost přidání bodu dvojklikem levým tlačítkem myši na hranu. Dalším mínusem je nemožnost navázání hrany na kterýkoliv bod okraje přechodu, hrana se automaticky přichytí na nejbližší možné místo. Jelikož u místa s výběrem bodu přichycení není problém, absence této možnosti u přechodu nedává příliš smysl.



Obrázek 5.1: příklad aplikace PNEditor - a Petri Net editor.

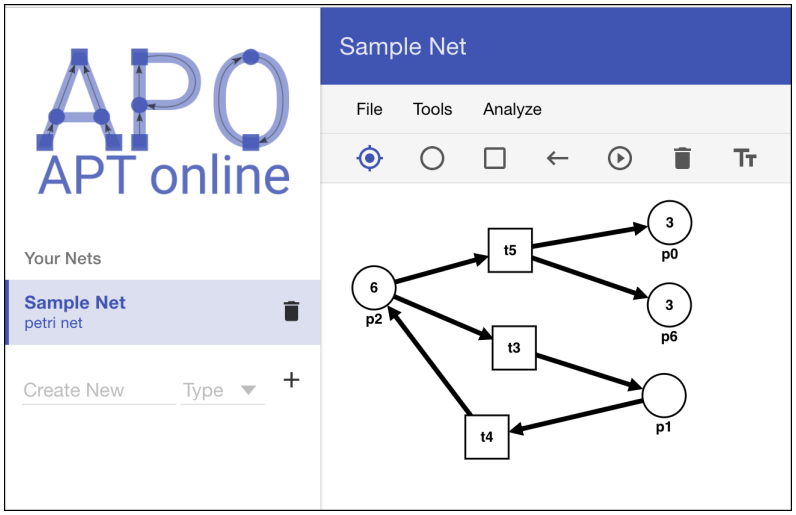
5.2.2 APO - APT online editor

Tento nástroj, napsaný v jazyce CoffeeScript[3], vznikl v Německu pro studijní účely. Patří k jednodušším, nelze v něm modelovat vysokoúrovňové Petriho sítě, avšak je zajímavý především svým příjemným, moderně vypadajícím GUI, jak můžeme vidět na obrázku 5.2 a skutečností, že se jedná o aplikaci spustitelnou přímo v prohlížeči, přičemž aplikace není ochuzena o možnost ukládání vytvořených modelů.

V liště si zvolíme nejen objekt sítě, který chceme přidat, ale také zde vybíráme akce, které chceme s objektem provádět. Například editace názvu neprobíhá přes kontextové menu jako v předchozím případě, ale výběrem editačního nástroje v liště a až poté kliknutím na požadovaný objekt. Tento přístup je zajímavý, avšak v našem nástroji se přikláním k provádění akcí s objektem přes kontextové menu. Mínusem této aplikace je bezesporu nemožnost vytvořit lomenou šipku.

Vidět různé přístupy k modelování Petriho sítí, bylo přínosné, v obou aplikacích bylo mnoho podmětů přínosných pro náš vývoj. Již po krátké době používání APT online editoru uživatel zjistí, že absence lomené šipky je dost nepříjemná už při vytváření Petriho sítí asi o 10 prvcích, a tak pokládám za důležité tento prvek do našeho nástroje implementovat. Dále bylo zjištěno, že je dosti užitečné mít možnost navázat hranu na kterýkoliv bod okraje

[3]CoffeeScript - skriptovací jazyk překládaný do JavaScriptu



Obrázek 5.2: příklad aplikace APO - APT online editor

prvku, obzvláště tomu tak bude v našem případě, kdy místa a přechody budou mít kvůli obsaženému textu často mnohem větší velikost.

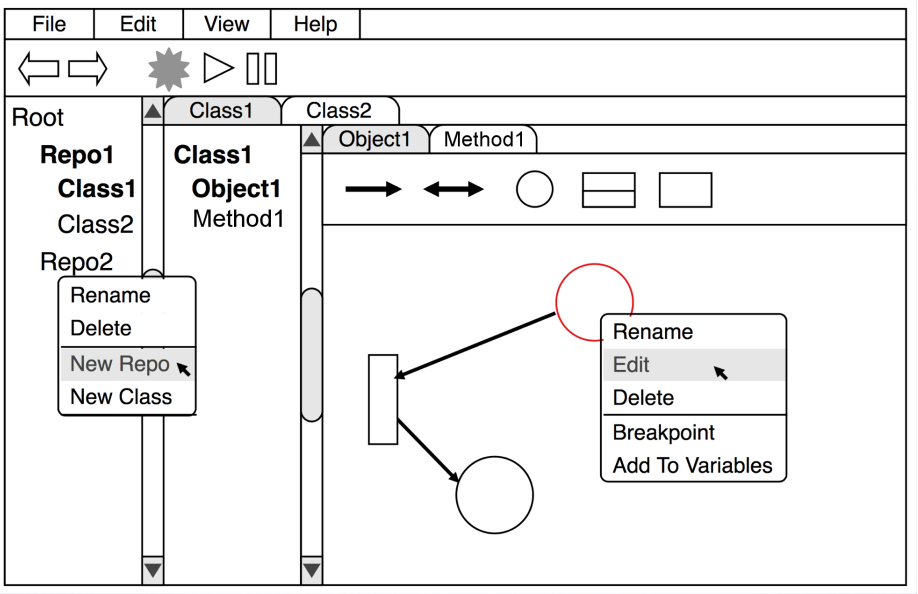
5.3 Analýza vývojových nástrojů

Program jsme se rozhodli implementovat v jazyce Java, a to především díky objektové orientovanosti jazyka a díky dobré podpoře pro tvorbu grafického rozhraní v knihovně Swing. Navíc kód jazyka Java je velice dobře přenositelný díky jeho běhu na virtuálním stroji. Na základě pozitivních minulých zkušeností bylo jako vývojové prostředí pro vývoj v jazyce Java použito IntelliJ IDEA od firmy JetBrains. Samotná simulace probíhá v jazyce Smalltalk na serveru v prostředí Pharo. Implementace serverové části již nebyla součástí projektu.

Kapitola 6

Návrh a implementace

V této kapitole bude představen jednak návrh GUI sestaveném na základě analýzy v předchozí kapitole a návrh architektury programu, včetně popisu balíků a příkladů použití objektově orientovaných přístupů. V další části přejdeme k popisu implementací jednotlivých částí programu. Popsány budou zejména části zajímavé z hlediska implementace, ale také tzv. slepé cesty, části, jejichž implementace byla jen uvažována nebo použita, ale posléze změněna. Lehce rozvedeny budou i další možné alternativní přístupy k řešení dané problematiky.



Obrázek 6.1: Grafický návrh aplikace.

6.1 Návrh grafického uživatelského rozhraní

Navrhování GUI bylo navázáno na již navrženou a implementovanou část doktorem Radkem Kočím, vedoucím práce. Ten navrhl základní rozložení panelů. Jak můžeme vidět na obrázku 6.1, v levé části byla dvakrát použita komponenta JTree, vlevo pro zobrazení adresářové

stromové struktury tříd a vpravo pro zobrazení objektových sítí a metod třídy. Největší část zabírá modelovací plocha tvořena JPanelem.

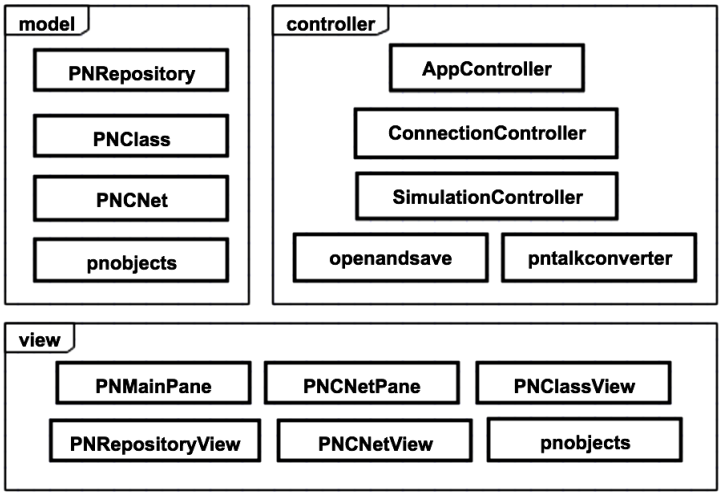
Na část navrženou doktorem Kočím bylo navázáno návrhem nástrojových lišt. Na základě analýzy v předchozí kapitole a konzultace s vedoucím práce jsem se rozhodl inspirovat editorem popsaným v sekci 5.2.1 v provádění většiny editačních akcí skrze pravé tlačítko. Z editoru ze sekce 5.2.2 jsem se inspiroval v použití velkých, jednoduchých, ale jasných ikon pro tlačítka lišt. Obě nástrojové lišty byly implementovány komponentou JToolbar. Editace textových řetězců a zobrazování výstražných hlášek probíhá v nově otevřených dialogových oknech.

6.2 Návrh architektury

Základ architektury byl podobně jako základ GUI navržen vedoucím práce. Navázáno bylo na hotovou adresářovou strukturu tříd a sítí a na vykreslovací panel se základní funkcionalitou.

6.2.1 MVC architektura

Snaha byla o implementaci MVC (model-view-controller) architektury. Ta od sebe odděluje datový model, zobrazovaný výstup a řídící logiku. Řeší tak problém tzv. špagetového kódu, kdy jsou tyto tři části provázány a výměna pouhého modelu nebo uživatelského rozhraní je kvůli nutným změnám v kódu dosti obtížná, v obzvláště špatných případech je nutné aplikaci navrhnout a implementovat zcela znovu. V našem případě byla tato architektura místy upravena, často při použití návrhového vzoru observer[1] v GUI, kdy řídící kód spuštěný změnou na grafické komponentě se ponechá ve třídě grafické komponenty. Tato technika se stále dost využívá kvůli neoddiskutovatelné logické návaznosti grafické komponenty s řídícím kódem.



Obrázek 6.2: Zjednodušená struktura hlavních tříd a balíků editoru.

[1] Observer - návrhový vzor, který umožňuje objektu ohlašovat změnu stavu všem objektům, které to vážadují, v praxi znám pod pojmem listener

6.2.2 Popis hlavních řídících tříd

Balík controller je složen z Appcontrolleru, který řídí běh aplikace. Z něj se volají například třídy balíku openandsave pro ukládání a načítání objektů. Balík ConnectionController se stará o spojení se vzdáleným serverem, je zodpovědný za odesílání zadaných příkazů a za naslouchání odpovědí. Samotné příkazy tomuto kontroleru předává třída SimulationController, která má na starosti řízení celé simulace. V třídách balíku pntalkconverter se spouští konverze modelů potřebné pro vzdálenou simulaci.

6.3 Třída

Třída (v kontextu OOPN) je implementována v třídě PNClass a PNClassView. Zde bylo důležité vyřešit problím dědičnosti. V případě změny v třídě se musí změny aplikovat také do tříd, které z této, třídy dědí. Stejně tak tato třída musí naslouchat změnám provedeným v nadtřídě. Tento problém je vyřešen implementováním listeneru, každá třída si udržuje v třídních proměnných seznam listenerů, které se v podtřídách spustí při každé změně v této třídě. Důležité bylo zamezit aplikaci určitých změn v případě, že podtřída přepisuje síť nadtřídy. U metod, které se v případě přepisování přepisují celé, se nastavuje příznak override, podle kterého metoda povolí její přepsání nebo nikoliv. U objektové sítě je situace o něco komplikovanější, protože je dovoleno přepisovat pouze části sítě objektu. Problém ale řeší samotný formalismus PNtalku, kde nemohou existovat v jedné síti stejné typy prvků stejného jména.

Každá síť má dvě vrstvy, jedna představuje vrstvu třídy, do které síť patří a druhá představuje síť nadtřídy. Zobrazování stavů jednotlivých instancí tříd za běhu simulace je provedeno přidáním samostatného modelu třídy pro každou instanci do stromové struktury třídy. Tyto instance jsou od sebe rozeznány unikátním ID v rámci instancí stejné třídy.

Názvy tříd na rozdíl od názvů komponent sítě nemohou obsahovat bílé znaky kvůli syntaxi jazyka PNtalk, kdy se název třídy právě za pomocí mezer a klíčového slova „is_a“ odděluje od její nadtřídy.

6.4 Síť

6.4.1 Model sítě

Implementaci sítí nalezneme ve třídách PNCMethodNet a PNCObjectNet, většina metod a třídních proměnných je společných a tak jsou implementovány v třídě PNCAbstractNet, která je třídou nadřazenou pro oba typy sítí. Jak již bylo zmíněno v předchozí podkapitole, každá síť je složena ze dvou vrstev, každá vrstva obsahuje hašovací mapu pro každý typ prvku, který se do mapy ukládá s jeho názvem jako klíčem. Při vyhledávání prvku v objektové síti podle názvu se prohledává nejprve vrstva představující třídu, do které síť patří, až poté vrstva patřící síti nadtřídy.

6.4.2 Vykreslování sítě

Modelovací plocha je tvořena třídou NetPanel, která dědí z komponenty JPanel. Samotné vykreslování objektů probíhá za pomocí Graphics2D. To je API[2], které umožňuje vykreslovat 2D vektorovou grafiku přímo do grafických komponent jazyka Java. Při každé změně

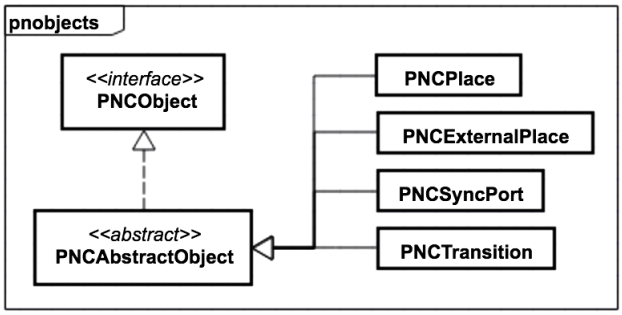
[2] Application Programming Interface - Aplikační rozhraní

v modelu (ten obsahuje informace i o grafické části), po které je třeba panel překreslit, se zavolá v třídě NetPanel metoda paint, z které se postupně volá metoda paint pro příslušné view třídy jednotlivých prvků.

Tohle API má jednu nevýhodu, ta spočívá v tom, že vektorové objekty nejsou konvertovány při zobrazování do rastrové grafiky. To způsobuje, že okraje vykreslených objektů jsou zubaté, protože jsou tvořeny pixely, které mají vždy stejnou hodnotu alfa kanálu[3]. Tento fakt není žádoucí, obzvláště na monitorech s nižším rozlišením jde rozpixelování obrazu hodně vidět.

6.5 Prvky Petriho sítě

Všechny modelové třídy prvků jsou obsaženy v balíku pnobjects. Na obrázku 6.3 můžeme vidět, že jednotlivé prvky sítě dědí z abstraktní třídy PNCAbstractObject, která implementuje rozhraní PNCObject. Použití abstraktní třídy a rozhraní umožňuje lehké rozšíření nástroje o další prvek. Další společný rys těchto tříd je, že jejich reprezentace obsahu je vytknuta do zvláštní třídy PNContent. V této třídě se mj. počítají rozměry obsahu prvku (rozměry obsaženého textu), na základě kterého se při změně obsahu vypočítají celkové rozměry prvku. Rozměry prvku jsou tak dynamicky závislé pouze na svém obsahu a to přináší výhodu, že uživatel nemusí velikost modelovaného prvku nastavovat ručně a vždy bude vidět všechen text při zachování největší možné úspory místa. Výhodou jednotné třídy pro obsah prvku je opět modifikovatelnost kódu. Díky dnešním pokročilým refaktorovacím metodám lze v případě rozšíření o jiný modelovací formalismus z abstraktní třídy PNCAbstractObject vytknout obecnější část kódu do nové abstraktní třídy z které bude původní třída dědit.



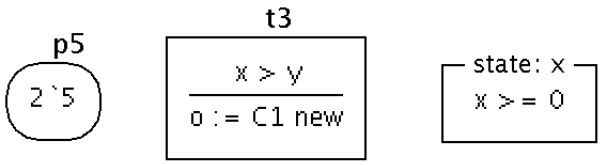
Obrázek 6.3: Struktura prvků v balíku pnobjects.

Grafická reprezentace míst je modelována za použití třídy RoundRect2D, to je třída z API Graphics2D, jejíž objekty jsou zobrazeny jako obdélník s kulatými rohy. Tento přístup je poněkud nezvyklý, většinou je zvykem místo modelovat jako elipsu v případě více textu, avšak rozhodl jsem se pro použití RoundRect2D z důvodu větší úspory místa, která jde u míst obsahujících mnoho různých proměnných na každém řádku opravdu znát.

Do třídních metod lze přidat místo z objektové sítě, to se modeluje pomocí třídy PNCExternalPlace, která obsahuje odkaz na modelované místo. Tyto modely je třeba uchovávat zvlášť z důvodu rozdílných hodnot třídních proměnných týkajících se grafiky a vykreslování

[3] hodnota určující průsvitnost pixelu

objektu. Externí místo je od míst patřících metodě odlišeno modrou barvou. View místa sítě a externího místa objektové metody představuje stejná třída PNCPlaceView. Grafická reprezentace Synchronizačního portu pro změnu dědí z třídy pro vykreslování přechodu PNCTransView.

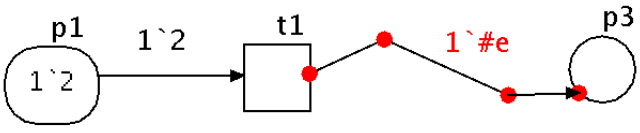


Obrázek 6.4: Příklad vymodelovaných prvků naším editorem, zleva: místo, přechod, synchronní port.

6.6 Hrany

Na základě zkušeností s předchozími aplikacemi bylo zjištěno, že vytváření hran mezi prvky je většinou nejčastěji prováděná činnost během modelování, a také, že proces modelování hran byla největší slabina obou zkoumaných aplikací. Jelikož uživatelské rozhraní má velký dopad na uživatelskou zkušenost s aplikací, rozhodl jsem se na modelování hran obzvláště zaměřit. Před samotnou implementací bylo vytyčeno pár bodů, které tvorba hrany musí splňovat:

- modelování jednoduché (nelomené) hrany pouhým kliknutím levým tlačítkem na příslušné prvky sítě (nikoliv na okraj prvku)
- možnost modelování lomené šipky, kterou lze navázat na kterékoliv místo okraje prvku
- jednoduchá editace šipky - zahrnuje možnost přiřazení konce šipky k jinému prvku sítě a vytváření bodu zlomu přes dvojklik levým tlačítkem myši



Obrázek 6.5: Příklad jednoduché a lomené šipky vymodelované naším editorem (lomená šipka je vybrána).

Všechny body nakonec byly implementovány za pomocí Path2D, což je třída z rodiny Graphics2D. Při zahájení modelování hrany se vytvoří instance třídy PNCArcBuilder, která je zodpovědná za tvorbu hrany a za udržení konzistentního stavu vytvářeného modelu. Například je nežádoucí, aby byl proces modelování hrany dokončen nenavázaným koncem na žádný prvek sítě, s tím souvisí fakt, že bylo také implementováno odstranění každé hrany, která je navázána na právě odstraňovaný objekt.

6.7 Ukládání modelů

Pro reálné využití nástroje je nutné vytvořit systém pro ukládání a načítání objektových sítí. K tomu je možné použít více přístupů. V našem nástroji byl tento systém realizován pomocí značkovacího jazyka XML[4]. V následující podkapitole si představíme různé způsoby, jak tuto problematiku řešit. V podkapitole 6.7.2 si popíšeme implementaci námi zvoleného způsobu.

6.7.1 Alternativní přístupy

První a také nejjednodušší možností je přímé použití serializace Java objektů, která se lehce zprovozní přidáním implementace rozhraní Serializable do příslušných tříd. Tohle řešení má ale značnou nevýhodu, a to že objekty jazyka Java se ukládají v binárním kódu, a tak není možné je číst a editovat. Právě proto byl zvolen značkovací jazyk XML, který editaci objektů umožňuje.

Existuje více způsobů ukládání dat do XML. Jednou z možností je použití knihovny JAXB (Java Architecture for XML Binding), která se sama postará o ukládání anotacemi označených proměnných do elementů a atributů jazyka XML. Tohle řešení nakonec nebylo použito kvůli žádné dosavadní zkušenosti s použitím knihovny na zpracování XML struktury polymorfní povahy (ta je zapříčiněna možností zanořování tříd a repozitářů do sebe).

Dále se nabízí použití jazyka PNML, což je jazyk založený na XML, který obsahuje standardy pro ukládání různých druhů petriho sítí. To má obrovskou výhodu díky možné přenositelnosti Petriho sítí mezi různými programy. Problém ovšem je, že pro koncept OOPN, neexistuje žádný standard, tzn. že o zmíněnou výhodu přenositelnosti tak přicházíme. Na základě toho jsme vyhodnotili, že nutné přidání dat do struktury XML pro splnění požadavků PNML postrádá smysl, pokud tato zvýšená režie nepřináší žádnou přidanou hodnotu. Jazyk PNML tak alespoň posloužil pro inspiraci k návrhu vlastní XML struktury.

6.7.2 Popis ukládání modelů v XML

Námi zvolený přístup je ukládání dat do jazyka XML za použití API DOM (Document Object Model). Tento balík obsahuje parser pro serializaci dat do XML a deserializaci z XML. DOM umožňuje také náhodný přístup k prvkům. Pro naši potřebu byly vytvořeny náležitě pojmenované třídy Serializer pro ukládání objektů do XML a Deserializer pro načítání dat z XML do Java objektů. Ukládání i načítání modelů bylo zrealizováno pomocí analýzy rekurzivním sestupem, která byla použita díky malému počtu jasných pravidel, která zpravidla přímo určují samotné identifikátory uzlů. K určení pravidla v uzlu XML není většinou potřeba znalost dalšího kontextu. DOM má oproti knihovně JAXB také výhodu, že je součástí Java API, takže není potřeba přidávat dodatečné knihovny. V příloze D najdeme příklad jednoduchého modelu uloženého v námi definované struktuře.

6.8 Simulace modelů

Kromě editační funkce tohoto nástroje je neméně podstatnou částí funkce simulační. Před popisem simulačního procesu je třeba popsat řešení zobrazování stavu jednotlivých instancí

[4] eXtensible Markup Language - značkovací jazyk pro ukládání a transport dat, při zachování čitelnosti člověkem

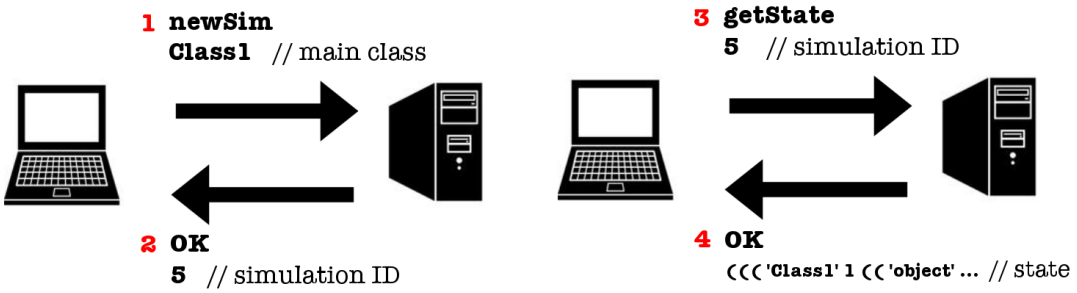
tříd, konverzi Java objektů do jazyka PNtalk a také představit proces aktualizace modelu novým stavem.

6.8.1 Konverze modelů z Java objektů do jazyka PNtalk

Nabízelo se zde použití konverze do PNtalku z modelu uloženého v jazyce XML, avšak k přidání tohoto mezikroku nebyl shledán žádný důvod. Jazyk XML je tedy generován přímo z již vytvořeného modelu v Javě. Za převod těchto modelů je zodpovědná třída PNtalkConverter. Třída v určitém pořadí zapisuje data z objektů do bufferu StringWriteru. Nástroj nezručuje správnost vygenerovaného kódu v PNtalku, nemá funkci syntaktického a sémantického analyzátoru, tuto funkci zastává server, který v případě chyby pošle příslušnou chybovou hlášku, která bude následně vytištěna do chybového dialogu.

6.8.2 Konverze stavu simulace do Java objektů

Po vykonání kroku simulace (popř. určitého počtu kroků) server pošle odpověď v které najdeme informaci, jestli se krok podařilo provést. V případě že ano, tato zpráva pokračuje popisem aktuálního stavu simulace. Je zapotřebí, aby si klient udržoval průběžný stav simulace. Na straně serveru je s tímto počítáno, odesílá klientovi pouze místa, jejichž obsah byl změněn v posledním kroku simulace. To ušetří velikost zasílaných zpráv, zejména u rozsáhlých modelovaných systémů.



Obrázek 6.6: Příklady komunikace klienta a serveru. zleva: nová simulace, dotázání stavu simulace.

Samotný stav simulace je strukturován do zanořených závorek. Pro parsování stavu byl zvolen konečný automat, který obsahuje stavy představující jednotlivá zanoření. Stav, ve kterém se analyzátor nachází určuje počítadlo state, které se při každé levé závorce na vstupu inkrementuje a při každé pravé závorce dekrementuje. Stav obsahuje v několika úrovních pár závorek, jejichž obsah je třeba ignorovat. Jelikož v rámci jedné úrovně mají závorkové seznamy určité pořadí, situace je ošetřena nastavením zámku v určité chvíle na příslušné stavy.

6.8.3 Spojení se serverem

Komunikaci se serverem zprostředkovává balík connection, v kterém se nachází třídy ConnectionController a IOUtil. Třída ConnectionController přímo odesílá zprávy na vzdálený

server za pomocí Apache Commons Net API. Tohle rozhraní nám zajišťuje komunikaci se serverem a tak se nemusíme zaobírat vytvářením spojení pomocí schránek. Třída IOUtils obsahuje buffer pro zachytávání vstupů a výstupů serveru.

6.8.4 Proces simulace

Všechny třídy podílející se na průběhu simulace se nachází v balíku simulation. Celý proces simulace je ovládán tlačítky v pravé části horní lišty. Před startem simulace se zadá v nastavení IP adresa serveru a číslo portu, v případě vynechání tohoto kroku se editor pokusí komunikovat se serverem na adrese localhost a na portu 9999. Simulace se spustí tlačítkem "new sim", jehož stisk postupně spustí konverzi modelu do PNtalku, odeslání příkazu addClass pro každou třídu a následné spuštění simulace na serveru Pharo. Simulaci je možné krokovat tlačítkem v horní liště po jednom nebo po více krocích.

Kapitola 7

Testování

Na začátku této kapitoly se zmíníme o jednotkových testech, v další části se budeme věnovat různým příkladům vytvořených i simulovaných naším editorem.

7.1 Jednotkové testování

Jednotlivé konvertory, pro serializaci objektů do jazyka XML, pro překlad Java objektů do jazyka PNtalk a pro konverzi stavu simulace, byly pokryty jednoduchými jednotkovými testy. Testy ani tak nesloužily k zaručení stoprocentní funkčnosti parserů, ale především k rychlé indikaci nutných oprav při změnách v modelu. Například při pozdější změně v Java objektech modelu sítě bylo ihned rozpoznáno, které části konvertorů je třeba pozměnit.

7.2 Testování na příkladech

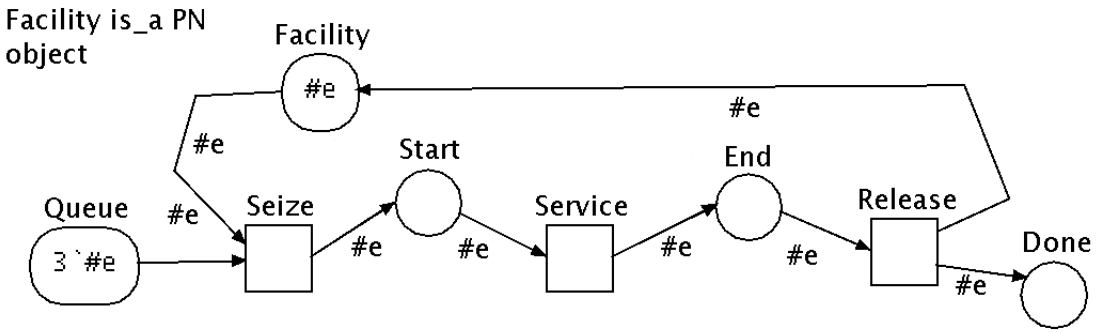
V této části si ukážeme náš editor na konkrétních příkladech. Každý příklad bude prezentován jednak náhledem stavu v nultém kroku a také tabulkou zobrazující stavy míst ve všech krocích simulace. Editor bude prezentován na příkladech s linkou s výlučným přístupem. V prvním případě v jednoduché verzi bez dalších rozšíření a v druhém případě s použitím dědičnosti, synchronního portu a tvorbou nových objektů.

Bylo zamýšleno uvést i příklad s metodou, ale aktuální verze aplikace spouštěné na serveru neumožňuje spouštět příklady s metodami, po prvním kroku simulace zůstane klient bez odpovědi.

7.2.1 Příklad jednoduchého zařízení

První příklad bude demonstrován na obslužné lince typu zařízení. Tato technika modeluje obsluhu zařízení s výlučným přístupem, kdy může v jeden čas probíhat pouze obsluha jednoho požadavku. [7]

Na začátku obslužné linky zobrazené na obrázku 7.1 se nachází místo Queue, které představuje frontu požadavků, čekajících na obsluhu. Jak můžeme vidět v tabulce 7.1, v krocích 1, 4 a 7 probíhá obsazování zařízení jednotlivými prvky. Názorně jde také vidět, že v jeden moment vždy probíhá obsluha pouze jednoho prvku.



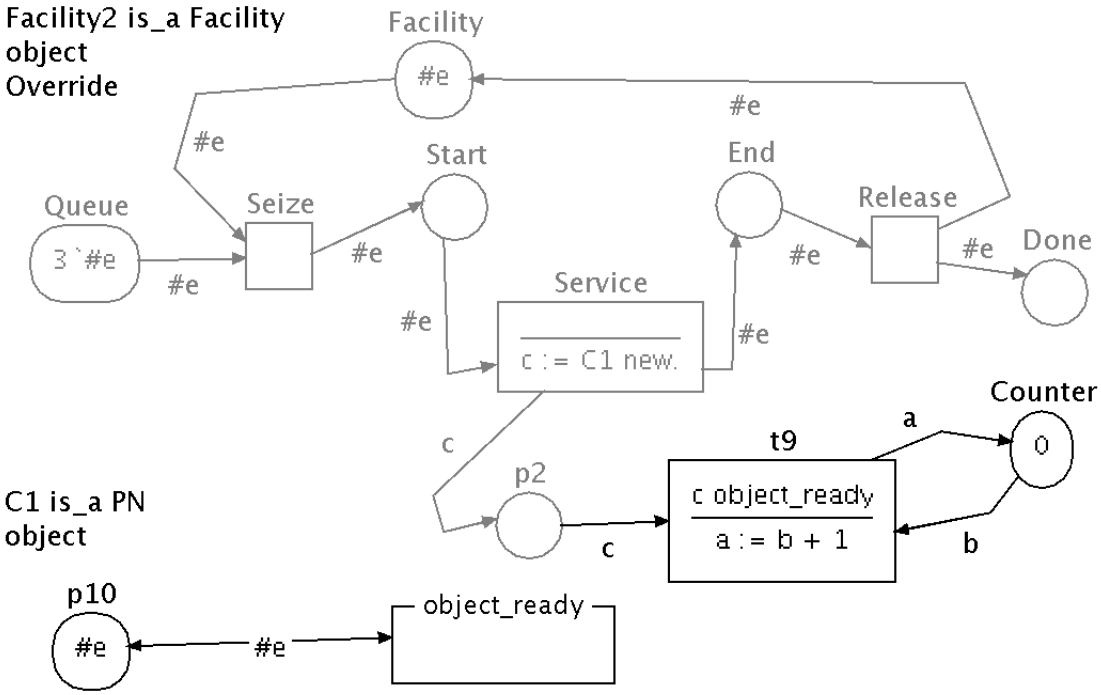
Obrázek 7.1: Příklad linky s výlučným přístupem.

Step	0	1	2	3	4	5	6	7	8	9
Facility	#e			#e			#e			
Queue	3‘#e	2‘#e	2‘#e	2‘#e	#e	#e	#e			
Start		#e			#e			#e		
End			#e			#e			#e	
Done				#e	#e	#e	2‘#e	2‘#e	2‘#e	3‘#e

Tabulka 7.1: Vyhodnocení míst sítě z obrázku 7.1 v jednotlivých krocích.

7.2.2 Petriho zařízení s počítadlem

Na následujícím příkladu si ukážeme navíc i použití dědičnosti, vytváření nových objektů a použití synchronního portu. V příkladu ze použije třída Facility z prvního příkladu obohacená o vytváření objektů v přechodu během obsluhy. Tohle zařízení tak může představovat výrobní linku vyrábějící určité objekty za použití prostředků z místa Queue. Z této linky vytváříme za použití dědičnosti třídu facility. Která představuje linku s počítáním vyrobených objektů. linka je obohacena oproti nadřazené třídě o přechod se stráží, v které se čeká na synchronizaci se synchronním portem, který je přítomen v třídě vyrobeného objektu C1 a indikuje například, že objekt byl vyroben úspěšně. V nově přidaném přechodu se s každým nově úspěšně vyrobeným objektem inkrementuje místo Counter představující počítadlo vyrobených objektů. V tabulce 7.2 si jen zobrazíme nově přidané stavy v nově vzniklé třídy Facility2, ostatní stavy nabývají v jednotlivých krocích stejných hodnot jako u sítě z předešlého příkladu.



Obrázek 7.2: Příklad výrobní linky s počítadlem a s vytvářením objektů nové třídy.

Step	0	1	2	3	4	5	6	7	8	9
p2			C1@2			C1@3			C1@4	
p10 C1@2			#e	#e	#e	#e	#e	#e	#e	#e
p10 C1@3						#e	#e	#e	#e	#e
p10 C1@4									#e	#e
Counter	0	0	0	1	1	1	2	2	2	3

Tabulka 7.2: Vyhodnocení míst sítě z obrázku 7.2 v jednotlivých krocích.

Kapitola 8

Závěr

8.1 Výsledek vývoje

Nástroj se podařilo implementovat do funkční podoby splňující všechny body zadání. Framework pro OOPN umožňuje ukládání i načítání modelů, které lze libovolně editovat. Editační část byla navržena se snahou o co nejpřívětivější rozhraní, ke kterému se bude koncový uživatel rád vracet, a které mu přinese co nejlepší uživatelskou zkušenost. Simulační část umožňuje verifikovat navržené modely a umožňuje použít všechny příkazy základní funkcionality. Snaha byla také o zachování modularity a o možnost rozšíření o další formalismy Petriho sítí. Rozšíření o implementaci nového nástroje pro zcela jiný specifikační formalismus je možné, ale tohle rozšíření se neobejde bez trochu větších zásahů do existujícího kódu, oproti přidání pouhého dalšího prvku OOPN.

Nástroj má však také svá omezení, vykreslování stavů instancí třídy při simulaci se někdy neaktualizuje hned, ale překreslí se stav až po kliknutí do vykreslovací plochy. Dalším omezením je, že momentálně nelze používat metody tříd, při spuštění simulace s jakoukoliv metodou se nedostává žádné odpovědi ze serveru.

8.2 Možná vylepšení

V této sekci si zmíníme několik možných vylepšení, která by bylo vhodné v budoucnu implementovat z důvodu rozšíření možností nástroje nebo zjednodušení ovladatelnosti. Některá ze zmíněných rozšíření byla v plánu, avšak z omezených časových možností k jejich implementaci již nedošlo.

- Operace undo a redo - uživatelé v dnešní době tento nástroj očekávají v každé obdobné aplikaci. Implementovat tyto operace bylo zamýšleno, avšak kvůli různorodým podobám akcí, pro které se tato operace použije se nejevila implementace jako zcela triviální a tak prozatím bylo od těchto operacích upuštěno.
- Klávesové zkratky - Bylo by vhodné v budoucnu implementovat více klávesových zkratek, zejména pro tvorbu prvků sítě. Ideálním řešením by byla možnost klávesové zkratky editovat přímo v GUI.
- Možnost zadání breakpointu zaregistrováním události v simulované síti - o zavedení tohoto rozšíření byla snaha, avšak při pokusu o zaregistrování události nepřicházela ze strany serveru žádná odpověď.

- Otevření posledního editovaného projektu při znovuspuštění aplikace.
- Větší syntaktická kontrola názvů tříd a prvků sítí.
- Možnost exportu vytvořeného modelu do jazyka PNtalk přímo z GUI.
- Možnost změny struktury v adresářovém stromu tak, aby byla tvořena hierarchicky podle dědičnosti.
- Možnost okamžité zobrazení sítě poděděného prvku, například přes stisk pravého tlačítka.

8.3 Metriky kódu a použité nástroje

Mezi použité nástroje patří vývojové prostředí IntelliJ IDEA zahrnující JDK verze 8, Pharo, prostředí pro spouštění simulací v jazyku Smalltalku na serveru, editor Atom pro validaci a upravování XML souborů. Program byl vyvíjen na platformě Apple Mac s operačním systémem macOS 10.13, testování proběhlo také na PC s operačním systémem Windows 10. Výhoda jazyka Java, přenositelnost mezi platformami, zůstala zachována.

- Počet souborů: 137
- Celkový počet řádků: 7071
- Velikost všech zdrojových souborů: 586 Kb

Literatura

[1] A. GOLDBERG, D. R.: *Smalltalk-80 the language*. Boston : Addison-Wesley, 1989, ISBN 0-201-13688-0, [cit. 2018-05-03].

[2] DORDA, M.: Úvod do Petriho sítí. [online] http://homel.vsb.cz/~dor028/Nekonvencni_metody_1.pdf, [cit. 2018-04-09].

[3] JANOUŠEK, V.: *Modelování objektů Petriho sítěmi*. Dizertační práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 1999, [cit. 2018-04-09].

[4] JANOUŠEK, V.; VOJNAR, T.: PNtalk Project. [online] http://www.fee.vutbr.cz/~janousek/pntalk/pntalk.html, 1997, [cit. 2018-04-09].

[5] KOČÍ, R.; JANOUŠEK, V.; ZBOŘIL, F.: Object Oriented Petri Nets - Modelling Techniques Case Study. ročník 10, č. 3, 2010: s. 32–44, ISSN 1473-8031. URL http://www.fit.vutbr.cz/research/view_pub.php?id=9194

[6] Milan ČEŠKA, V. M.; NOVOSAD, P.: Petriho Sítě. [online] http://www.fit.vutbr.cz/study/DP/PD.php?id=124, 2012, [cit. 2018-04-09].

[7] PERINGER, P.: Modelování a Simulace. [online] https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIMS-IT%2Ftexts%2Fopora-ims.pdf&cid=12167, 2012, [cit. 2018-04-09].

Seznam příloh

Příloha A

Obsah CD

- `editor/` - adresář obsahující zdrojové soubory a spustitelný jar soubor
- `server/` - adresář obsahující server
- `example/` - adresář s uloženým příkladem, který je možné editorem načíst
- `xmatal01.pdf` - technická zpráva
- `poster.png` - plakát
- `README` - návod pro spuštění editoru
- `LICENCE`

Příloha B

Manuál pro spuštění

- Nainstalujte si aplikaci Pharo verze 5.0 z webu http://pharo.org/web/download.
- Spusťte server z přiloženého CD server/PNtalk-2.1-server.image a po spuštění označte text PNtalkServer start, na něj pravým tlařítkem a klikněte na "Do it". Nyní by měl být server spuštěn.
- Spusťte editor z přiloženého CD editor/PNEditor.jar standardním příkazem pro spoužtění balíků jar. Není třeba zadávat cestu k třídě.
- Nyní můžete načíst příklad uloženého souboru ze složky /example.
- Před spuštěním simulace stiskem tlačítka "newsim"je třeba definovat hlavní třídu - to se provede otevřením kontextového menu pravým tlačítkem na uzlu v třídy v jejím náhledu a kliknutím na "Set as Main Class".

Příloha C

PNtalk

Příklad Petriho sítě z příkladu v podkapitole 7.2.2

```
class Facility is_a PN
  object
    place Queue(3‘#e)
    place Start()
    place End()
    place Done()
    place Facility(1‘#e)
    place p13()
    trans Seize
      precond Queue(1‘#e), Facility(1‘#e)
      postcond Start(1‘#e)
    trans Service
      precond Start(1‘#e)
      action {
        c := C1 new .
       }
      postcond End(1‘#e), p13(1‘c)
    trans Release
      precond End(1‘#e)
      postcond Done(1‘#e), Facility(1‘#e)

class Facility2 is_a Facility
  object
    place Counter()
    trans t7
      precond p13(1‘c)
      guard {
        c object_ready .
       }
      postcond Counter(1‘1)
```

Příloha D

Příklad uloženého projektu v XML

```
<?xml version="1.0" encoding="UTF-8"?>
<repo name="New Project">
  <class main="true" name="Facility2" parent="Facility">
    <objectNet name="object">
      <arc ID="26" superArcID="" type="ARROW">
        <text value="c" x="341" y="229"/>
        <from name="p13" x="30" y="13"/>
        <to name="t7" x="-1" y="20"/>
        <pointList/>
      </arc>
      <arc ID="27" superArcID="" type="ARROW">
        <text value="1" x="473" y="232"/>
        <from name="t7" x="105" y="23"/>
        <to name="Counter" x="0" y="10"/>
        <pointList/>
      </arc>
      <place contentType="CONTENT" name="Counter">
        <position x="492" y="220"/>
        <textUpper/>
        <textLower/>
      </place>
      <trans contentType="CONTENT_WITH_LINE" name="t7">
        <position x="357" y="210"/>
        <textUpper>c object_ready</textUpper>
        <textLower/>
      </trans>
    </objectNet>
  </class>
</repo>
```